UAB

Treball de Fi de Grau

Grau en Enginyeria de Sistemes de Telecomunicació

Implementació del protocol Galileo OSNMA

Aleix Galan Figueras

Director: Gonzalo Seco Granados

Departament de Telecommunicació i Enginyeria de Sistemes

Universitat Autònoma de Barcelona

Bellaterra, Setembre 01, 2020

Abstract

With the rise of GNSS based applications, there has been an increasing need for signal authentication features. A signal is considered authentic when the receiver can assure that the sender is a satellite of the GNSS constellation. Based on this idea, Galileo will include an authentication protocol for the E1 I/NAV message called Open Service Navigation Message Authentication (OSNMA). This report details the development of a Python package that implements the functionalities and features of the OSNMA protocol. The package is distributed publicly on the PyPI repository with its documentation and code. This report also describes the development of a receiver that implements OSNMA using the referred package that process real Galileo navigation data generated on an external simulator. The receiver shows to the user how OSNMA protocol works in a configurable, clear and simple manner.

Resum

Amb l'augment de l'ús d'aplicacions que utilitzen sistemes GNSS, s'ha incrementat la demanada de mecanismes per assegurar l'autenticitat del senyal. Per autenticitat del senyal s'entén la seguretat que té el receptor que la senyal rebuda prové d'un satèl·lit de la constel·lació de GNSS que esta emprant. En aquesta línia, i per assegurar l'autenticitat del senyal, Galileo està implementant un protocol d'autenticació anomenat Open Service Navigation Message Authentication (OSNMA) en el missatge I/NAV E1. Aquest informe detalla el desenvolupament d'un paquet públic de Python que implementi les funcionalitats i característiques del protocol OSNMA. Del paquet desenvolupat s'ofereix el codi i la documentació per facilitar-ne implementacions futures. L'informe també presenta el desenvolupament d'un receptor que implementa OSNMA fent servir el paquet dissenyat i que processa dades reals de Galileo generades per simulador extern. El receptor mostra de manera clara el funcionament intern del protocol OSNMA.

Índex

Índex de figures

Capítol 1

Introducció

L'increment de dispositius que utilitzen els sistemes GNSS (*Global Navigation Satellite System*) per posicionar-se fa cada vegada més rellevant assegurar que el senyal d'aquests no es veu manipulada per agents externs. Els sistemes GNSS funcionen enviant un senyal en *broadcast* des d'una flota de satèl·lits. El senyal està modulat amb un Coarse Acquisition Code que és conegut pel receptor i que mitjançant una correlació del senyal rebut amb el codi propi generat permet ajustar l'antena per enganxar-la al senyal. Quan el receptor està enganxat al senyal, sabent la velocitat de propagació del senyal i la posició dels satèl·lits (n'ha de tenir 4 a la vista per a poder aïllar les tres variables de posició i el temps) pot determinar la seva posició, la velocitat i el temps actual. Per saber la posició dels satèl·lits la pot obtenir de les dades transmeses pel senyal rebut, així com altra informació rellevant de la transmissió [Mis06].

El receptor depèn del senyal rebut per calcular la seva posició, velocitat i temps. Això fa que, si un atacant volgués induir a errors al receptor, podria enviar un senyal que el receptor interpretes com a autèntic i enganyar-lo. Aquesta acció és el que s'anomena *Spoofing* i és especialment perillosa per receptors situats a components crítics.

Existeixen diverses tècniques d'*spoofing*, a la secció 1.1 s'explicaran de manera sintetitzada les més rellevants i a la secció 1.2 alguns mecanismes de defensa per detectar-lo i evitar-lo. A la secció 1.3 es detallarà el funcionament del protocol OSNMA que vol implementar el sistema Galileo per evitar l'*spoofing*.

Un cop repassat el marc teòric, al capítol 2 es definirà l'objectiu del projecte així com l'estat de l'art en aquest camp. Al capítol 3 s'explicarà la metodologia a seguir per dur a terme el projecte. Al capítol 4 s'exposaran les dificultats, decisions i fases del desenvolupament del projecte. Al capítol 5 es mostraran els resultats obtinguts. Finalment, al capítol 6 es presenten les conclusions que tanquen aquest document.

1.1 Atacs d'*spoofing*

Abans de veure com diferents atacs d'*spoofing* intenten replicar el senyal original, cal saber quina forma té aquest senyal. Per aquest propòsit ens basarem en la versió genèrica proposada per [Psi16] o [Joh17]:

$$y(t) = \mathrm{Re}\left\{\sum_{i=1}^{N} A_i D_i[t-\tau_i(t)]C_i[t-\tau_i(t)]e^{j[\omega_c t-\phi_i(t)]}\right\}$$

On N és el nombre de senyals del codi d'expansió, A_i és l'amplitud de l'i-é senyal portador, $D_i(t)$ és l'i-é senyal de dades, $C_i(t)$ és el codi d'expansió, $\tau_i(t)$ és la fase del codi de l'i-é senyal, ω_c és la freqüència nominal de la portadora i $\phi_i(t)$ és la fase de la portadora. El senyal que envia un *spoofer* intenta imitar alguns d'aquests paràmetres:

$$y(t) = \mathrm{Re}\left\{\sum_{i=1}^{N} A_{si}\hat{D}_i[t-\tau_{si}(t)]C_i[t-\tau_{si}(t)]e^{j[\omega_c t-\phi_{si}(t)]}\right\}$$

Òbviament el codi d'expansió $C_i(t)$ ha de ser el mateix que el senyal que es vol imitar, ja que a més és conegut. Les dades $\hat{D}_i(t)$, en canvi, son un paràmetre que s'ha d'intentar estimar. Els valors A_{si}, $\tau_{si}(t)$ i $\phi_{si}(t)$ són valors que l'*spoofer* pot voler modificar arbitràriament per perpetrar l'atac. Cal recordar que el receptor de la víctima rebrà alhora el senyal original i el senyal fals, junt amb soroll.

El primer que ha de fer un *spoofer*, doncs, és fer que el receptor s'enganxi al nou senyal i en general hi ha dues maneres d'aconseguir-ho. La primera és provocar interferències al receptor de manera que perdi el seguiment del senyal normal i intenti tornar-lo a aconseguir. En aquest moment es transmet el senyal fals amb una amplitud molt més gran que el normal ($A_{si} >> A_i$) per provocar que el receptor s'enganxi al fals. La segona manera consisteix a transmetre el senyal fals totalment alineat en fase ($\tau_{si}(t) = \tau_i(t)$) amb el senyal normal a l'antena del receptor. Llavors es comença a incrementar l'amplitud del senyal fals fins que rabassa la potència del senyal normal i en aquest punt, amb el receptor enganxat al nou senyal, es procedeix a canviar la fase de manera coordinada (*drag-off*) per induir a càlculs erronis a l'objectiu.

L'atac d'*spoofing* en el que és més fàcil generar el senyal fictici (és a dir, els valors de $C_i(t)$ i $D_i(t)$) s'anomena *meaconing*. El *meaconing* consisteix en enregistrar el senyal real i immediatament tornar a transmetre'l amb un guany suficient per enganyar al receptor. Aquest mètode funciona fins i tot si el codi $C_i(t)$ és desconegut, ja que no modifica les dades del senyal. Ara bé, la posició que rebrà el receptor serà la del receptor de l'*spoofer* i un temps inferior al temps real [Joh17].

Si el senyal conté parts impredictibles a les dades $D_i(t)$ i varien lentament, l'atacant pot

intentar estimar els valors i fer un atac d'*spoofing* sense *meaconing*. Aquest atac s'anomena *Security Code Estimation and Replay* (SCER) [Hum13]. Aquest segon atac necessita temps de còmput extra per realitzar l'estimació i si triga molt el senyal que s'enviï estarà fora de la desviació de temps acceptable pel rellotge del receptor atacat.

Finalment, un tercer tipus d'atac anomenat *nulling* envia 2 senyals: el senyal *spoofer* i el senyal verdader amb un desplaçament en fase de 180º, que generi una interferència destructiva amb el senyal del satèl·lit i l'anul·li. Tot i que el concepte és senzill, implica modificar i calibrar molts components físics i no és trivial de dur a terme [Psi16].

1.2 Defenses davant l'*spoofing*

Per protegir-se davant dels diferents atacs esmentats també existeixen varietat de tècniques. La primera és detectar l'inici de l'atac: increments anormals de l'amplitud o error en l'ajust de la fase de la portadora durant l'inici del *drag-off*. També es poden realitzar cerques periòdiques de senyals encara que el receptor ja estigui enganxat a un per detectar si s'ha estat desplaçat cap a un de fals.

Una altra defensa és mesurar els canvis en la posició i el valor del temps. Canvis molt ràpids i no raonables per la situació del receptor poden indicar que s'està rebent un senyal fals i activar sistemes per tornar a buscar el senyal. Aquest llindar indicant que es consideren canvis ràpids o impossibles pot variar per cada receptor i es podria programar o calcular-se en funció dels canvis anteriors amb una petita intel·ligència artificial.

Finalment, també existeix una tècnica anomenada *Navigation Message Authentication* (NMA) basada en un esquema d'encriptació asimètrica. El missatge signat es transmet en les dades del satèl·lit i, si el receptor coneix la clau pública, pot verificar que les dades rebudes només les ha pogut generar i signar el satèl·lit a qui correspon el parell de claus. Aquest mètode introdueix latència ja que necessita rebre la signatura completa per poder-la verificar i detectar un cas d'*spoofing* i pot ser vulnerable a errors durant la transmissió. També introdueix aleatorietat al missatge, fet que dificulta els atacs d'SCER. Per contra cal variar l'estructura de les dades que transmet el senyal GNSS per encabir els nous bits del protocol [Psi16]. En aquesta línia Galileo vol incorporar un sistema d'NMA en la pròxima generació, i en aquest sistema és en el que es centrarà la resta de l'informe.

1.3 OSNMA

El sistema NMA que incorporarà Galileo s'anomena *Open Service Navigation Message Authentication* (OSNMA), va ser proposat per [FH16a] i especificat en la versió 1.0 a [FH16b].

Actualment s'han publicat ampliacions a la versió 1.0 a l'article [Fer19]. En aquest informe, quan es faci referència al protocol OSNMA, serà a la versió 1.0 més les ampliacions proposades. El protocol OSNMA es vol implementar al missatge I/NAV de Galileo de la banda E1-B aprofitant els bits del camp 'Reserved 1' de la definició del senyal [Eur16]. L' I/NAV *subframe* es transmet 30s i conté tots els paràmetres necessaris per al posicionament excepte l'almanac, que es transmet de manera distribuïda entre diferents *subframes*. Cada *subframe*, alhora, està format per 15 pàgines de 2 segons de durada transmeses en blocs d'1 segon. Les pàgines disposen de 240 bits transmesos cada 2s, el camp 'Reserved 1' ocupa 40 bits que són transmesos cada 2 segons. En total, el protocol OSNMA disposa de 600 bits cada *subframe*, que el considerarem la unitat bàsica de transmissió.

1.3.1 Criptografia

Criptogràficament, el protocol OSNMA es basa en dos factors:

- El receptor disposa d'una clau pública que ha obtingut de manera fiable que li permet verificar altres claus mitjançant signatura digital ECDSA [Nat13].
- La clau verificada és la clau arrel del protocol TESLA [Per02] i permet verificar altres claus que es transmeten junt amb un *Message Autentication Code* (MAC) a la que proporcionen autenticació.

El protocol TESLA genera una cadena de claus a partir d'una clau llavor secreta (K_n) mitjançant una funció uni-direccional i no reversible, com per exemple una funció de resum (*hash*). L'última clau de la cadena s'anomena clau arrel (K_0) i es el resultat d'aplicar la funció resum n vegades de manera recursiva a la clau llavor. Aquesta cadena llavors es transmet de manera inversa a com ha estat generada, és a dir, primer K_0 i per últim K_{n-1}. D'aquesta manera, si un receptor normal intercepta una clau de la cadena K_m, pot comprovar que és autentica aplicant la funció de resum de manera recursiva m vegades fins a arribar a K_0. Comparant la clau arrel calculada amb la clau K_0 que té guardada pot determinar si la clau K_m rebuda és autentica. El receptor sap que K_0 és autentica, ja que l'ha rebut signada d'algú de qui confia tenir la clau pública.

En canvi, si un atacant volgués generar alguna clau que es transmetrà en el futur a partir d'una clau rebuda per crear un missatge d'*spoofing*, no podria fer-ho mentre la funció criptogràfica sigui segura i no-reversible. El que sí que podria fer un atacant és tenir precomputades diferents cadenes de *hash* i al detectar una clau pertanyent a una cadena sabria directament les següents claus. Aquest atac s'anomena de diccionari precomputat i per evitar-lo OSNMA especifica que la funció *hash* per crear les següents claus de la cadena no només faci un resum de la clau anterior

sinó de la concatenació del Galileo Satellite Time (GST) del *subframe* en què s'envia la clau i la clau. Aquesta tècnica és similar a la que s'utilitza per evitar atacs de diccionari precomputat en emmagatzemar contrasenyes afegint *salt* al *hash* de la contrasenya.

Tots els satèl·lits que implementen OSNMA fan servir la mateixa cadena de claus, però transmeten claus diferents cada un. Si transmetessin tots la mateixa clau, només la primera rebuda seria impredictible. Però al formar part de la mateixa cadena, es permet al receptor aconseguir informació d'altres satèl·lits per ajudar-se en cas que rebi un senyal degradat.

Les claus TESLA verifiquen per un procés HMAC [Nat08] les MACs transmeses. Aquestes MACs es transmeten junt amb les claus i fan referència a diferents dades del missatge I/NAV. També poden fer referència a dades de missatges d'altres satèl·lits o altres sistemes GNSS, permetent així l'autenticació creuada.

Quan una cadena TESLA arriba al final o ha estat compromesa, cal enviar una nova clau arrel K_0 per la següent cadena. En aquest punt també es poden canviar paràmetres de configuració de la nova cadena (mida de les claus, funcions utilitzades, mida de les MAC, etc). La nova K_0 es verifica amb la nova signatura digital transmesa i la clau pública guardada al receptor.

També pot ser que sigui la clau pública la que hagi estat compromesa o que es vulgui canviar periòdicament. Per aquest procés es fa servir un arbre de Merkle (Merkle Tree [Mer79]). El receptor ha obtingut de manera fiable i confia en l'arrel de l'arbre que té guardada. En les especificacions actuals, l'arbre de Merkle utilitzar té 16 fulles, és a dir, 16 possibles claus publiques que el receptor pot autenticar gràcies a l'arrel de l'arbre. L'ús d'un arbre de Merkle és especialment interessant en aquest cas, ja que transmetent només 4 nodes ($\log_2 num\_fulles$) per qualsevol de les fulles es pot autenticar la nova clau. A més, aquests nodes intermedis no presenten cap relació que permeti calcular altres claus com era el cas d'una cadena TESLA.

1.3.2 Estructura del missatge

El missatge d'OSNMA es transmet en blocs de 40 bits a cada pàgina, el que fa un total de 600 bits disponibles per cada *subframe* del missatge I/NAV. Cada 30s es transmet un *subframe* sencer i el protocol OSMA funciona transmetent unitats lògiques de dades coherents dins d'aquests *subframe*. Això no vol dir que no puguin haver missatges transmeses en més d'un *subframe*, sinó que cada *subframe* contindrà dades suficients per indicar la posició i lògica d'aquestes dades.

La implementació del protocol divideix els 600 bits per *subframe* en 2 missatges diferents que es transmeten paral·lelament com mostra la Figura 1.1. Cada missatge té les següents característiques:

- **HKROOT**: Ocupa 120 bits per *subframe* (8 bits per pàgina). Aquest missatge gestiona

la configuració del protocol OSNMA i transmet la clau arrel (K_0) i la clau pública actual. A cada *subframe* envia sempre una capçalera indicant el missatge que conté, ja que les claus ocupen més d'un *subframe*.

- **MACK**: Ocupa 480 bits per *subframe* (32 bits per pàgina). Aquest missatge transmet MACs i claus TESLA per verificar les dades. Sempre encaixa en els 480 bits del *subframe*.

El missatge HKROOT transmet sempre 2 capçaleres. La primera s'anomena NMA Header i indica si s'està fent servir el protocol, quina cadena cal fer servir per verificar les claus del *subframe* actual i si s'està duent a terme algun canvi de clau. La segona capçalera s'anomena DSM Header i indica la ID del missatge DMS transmès en els bits restants del *subframe* i la ID del bloc concret del missatge DSM transmès en aquest *subframe*. Aquests caps son necessaris perquè el receptor pugui reconstruir el missatge DSM a partir de diferents *subframes* que pot haver rebut de satèl·lits diferents, ja que es fa servir la mateixa clau.

El missatge DSM que transmet el HKROOT pot ser el DMS-KROOT o el DMS-PKR. El DMS-KROOT transmet una nova clau arrel per la cadena del protocol TESLA. A més de la clau, el missatge transmet opcions de configuració d'aquesta cadena (funcions a fer servir, mia de les claus i de les MACs, etc) i la signatura digital de la clau per poder-la verificar. El DMS-PRK transmet una nova clau pública, junt amb l'algorisme per la verificació i els paràmetres de l'arbre Merkle necessaris per verificar-la (4 nodes intermedis donat que l'arbre té 16 fulles).

Per una informació més detallada del protocol OSNMA i el seu funcionament es recomana la literatura [Fer19] [FH16a] [FH16b].

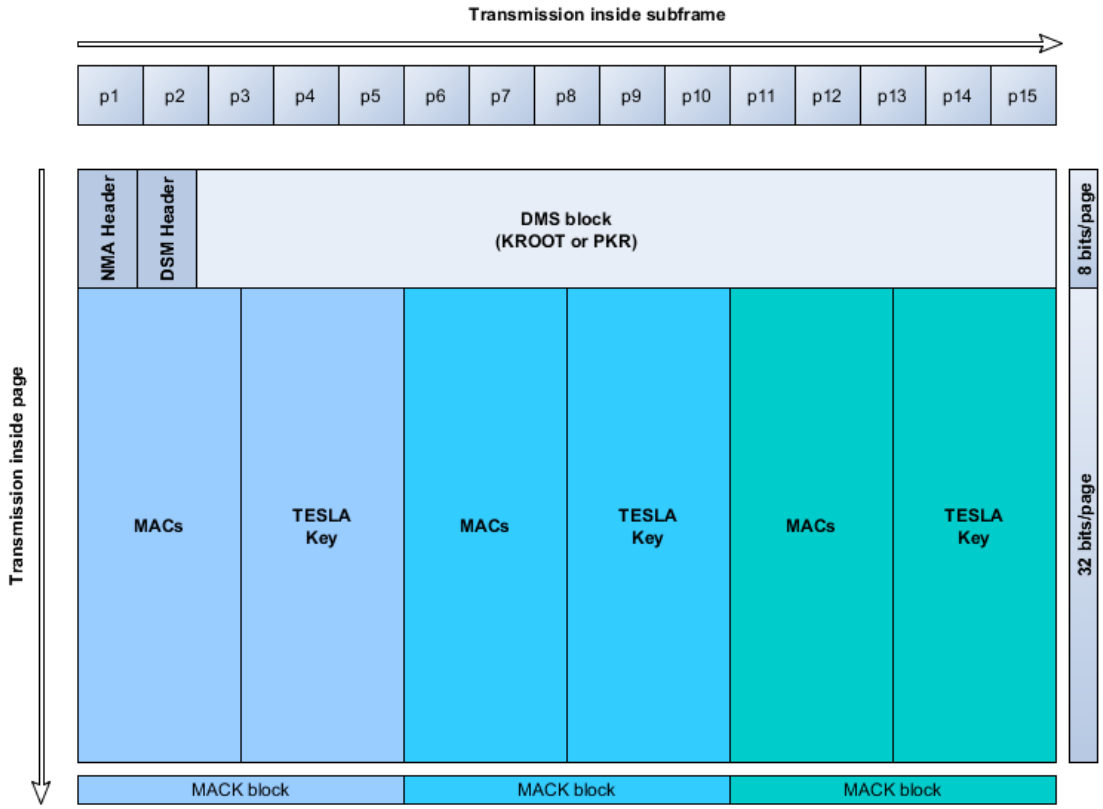


Figura 1.1: Estructura del missatge OSNMA a cada *subframe*.

Capítol 2

Objectius

2.1 Objectius del projecte

Els objectius del projecte són crear un paquet en Python 3 [VR09] que implementi el protocol OSMNA en la versió 1.0 [FH16b] afegint les ampliacions de l'article [Fer19], tant l'estructura de dades com les funcions necessàries, i fer-lo servir per crear un receptor que processi les dades del missatge E1-B I/NAV que implementa OSMA i mostrar el seu funcionament.

El paquet de Python 3 que implementa les operacions d'OSNMA ha de ser independent del receptor on s'utilitzi i ha de proporcionar una estructura d'emmagatzemat i mètodes que permetin realitzar les següents operacions d'OSNMA:

- Autenticar una nova clau pública.
- Autenticar una clau arrel (K_0).
- Autenticar una clau TESLA de la cadena.
- Autenticar les dades de navegació mitjançant HMAC.

A més, la llibreria ha de permetre realitzar *logging* de variables rellevants (a ser possible en format CSV). També ha de disposar de documentació per al seu ús i ser modular i orientada a objecte per les futures possibles modificacions. Finalment, haurà de complir amb el llibre d'estil oficial de Python: PEP 8 [Ros01].

El receptor programat fent servir el paquet ha d'implementar la lògica d'un receptor Galileo per tractar les dades d'I/NAV i adaptar-les al format requerit per la llibreria. També ha de permetre mostrar a l'usuari de manera informativa i divulgativa les operacions que s'estan realitzant, si pot ser mitjançant una interfície gràfica.

També són objectius del projecte: aprendre a programar en Python orientat a objecte i documentat, entendre el funcionament dels sistemes GNSS així com les tècniques i defenses davant l'*spoofing*, comprendre el funcionament del protocol OSNMA i fer servir un repositori Git per desenvolupar i seguir els canvis en el software.

2.2 Estat de l'art

S'han començat a implementar solucions d'OSNMA sobre receptors ja existents per avaluar paràmetres com el temps fins la primera posició autenticada (TTFAF), la velocitat a la que s'utilitzen claus i el temps que triga quan ha de canviar de claus. Per exemple [Zub18] mostra els resultats d'una implementació a tres receptors amb les diferencies de TTFAF entre l'opció del protocol que fa servir claus més curtes i aquella que fa servir les més llargues. També incideix en quin percentatge de la senyal pot arribar corrupta al receptor i tot i aixi reconstruir el missatge d'OSNMA a partir dels blocs d'altres satèl·lits.

En canvi l'article [Mot20] implementa el protocol OSNMA sobre el receptor ja existent NGene [Mol09] en un ordinador i avalua les necessitat de comput i memòria extres necessàries per l'ús OSNMA. A més, les proves es realitzen en temps reals amb les dades dels satèl·lits rebudes carregant-les a l'ordinador a partir d'un adaptador. Fragmenta l'anàlisi en les funcions primaries del protocol (verificar la clau TESLA, verificar la MAC, verificar la clau arrel i verificar una nova clau publica) i mostra els cops que es criden i el percentatge de temps respecte al total que consumeixen. Finalment, determinen que el cost computacional afegit per OSNMA està dins dels marges requerits per un receptor estàndard de Galileo. També ofereixen recomanacions per futures implementacions a fi d'evitar càlculs innecessaris i prevenir possibles atacs d'*spoofing*.

Totes elles, però, són implementacions tancades que no ofereixen el codi emprat en la implementació ni la documentació. Tampoc busquen mostrar al públic general el funcionament d'OSNMA.

Capítol 3

Metodologia

La metodologia per desenvolupar el projecte es divideix en dues fases que es realitzaran de manera seqüencial: una teòrica i una de pràctica.

3.1 Teoria

La fase teòrica serà la primera que es realitzarà i implicara conèixer el funcionament dels sistemes GNSS i en concret de Galileo, dels quals es té un coneixement nul. Serà necessari conèixer com un receptor pot determinar la seva posició i temps i el funcionament intern del senyal. Aquest coneixement es necessari per tenir una basi i saber per a que s'està desenvolupant.

A continuació, es realitzarà un estudi de les tècniques d'*spoofing* actuals sobre els sistemes GNSS i les possibles defenses. L'estudi ha d'entrar en quins són aquests atacs i el seu funcionament, que serà possible d'entendre ja que es té una base dels sistemes GNSS. També s'estudiaran les possibles defenses i quins atacs eviten, fent especial èmfasi a les defenses NMA.

Llavors, s'aprofundirà en el funcionament del protocols OSNMA i en el funcionament de totes aquelles operacions criptogràfiques necessàries per dur-lo a terme. Aquestes són: criptografia de clau pública i ECDSA [Nat13], funcions resum (*hash*) i l'estàndard SHA2 [Nat12] i SHA3 [Nat15], *Message Autentication Code* i TESLA [Per02].

També serà necessari conèixer l'estructura dels missatges I/NAV de Galileo definits per [Eur16] ja que caldrà que el receptor reconstrueixi el missatges i aïlli els bits corresponents al protocol i també aquells bits de dades que el protocol autentica.

Finalment, caldrà aprofundir el llenguatge de programació Python 3 [VR09] del qual es té un coneixement bàsic de la versió 2.7 però no s'ha programat orientat a objecte. Tampoc s'ha programat mai seguint les normés d'estil d'aquest llenguatge ni intentat fer un paquet, fet que

requerira més estudi. També s'haurà d'estudiar com documentar el codi a partir de comentaris en aquest.

3.2 Pràctica

Pel desenvolupament de la part pràctica es seguirà una metodologia incremental amb cicles de planificació, programació i test. La metodologia incremental permetrà assegurar el funcionament de parts més petites abans de seguir programant parts més grans. Com que es un projecte gran i nou, seguir aquesta metodologia permet avançar i tenir sempre disponible una versió del producte, en cas que sorgeixin dificultats i no es pugui acabar el projecte a temps. Per seguir els canvis al projecte s'utilitzarà la tecnologia *git*.

En un primer moment s'analitzarà com implementar les funcions de criptografia bàsiques necessàries pel protocol en Python 3.7 i es testejarà el seu funcionament per veure que compleixen les especificacions establertes a [FH16b]. A continuació es valorarà en quin format s'hauran d'emmagatzemar les dades que utilitza el protocol OSNMA per configurar-se i realitzar les operacions.

Un cop decidides les funcions criptogràfiques i el model de dades, es programaran les funcions principals del protocol de manera individual i es verificarà el seu funcionament. Per verificar que s'han programat bé s'utilitzaran les dades i vectors de test proporcionats a [FH16b].

A continuació s'implementaran aquestes funcions en el mateix objecte Python que en el futur serà el paquet proporcionat. I es tornarà a verificar el seu funcionament amb els vectors de test. També s'haurà de crear una manera de carregar i obtenir dades a l'objecte.

En aquest punt i amb la primera versió de la llibreria, es programarà el receptor OSNMA que incorporarà la lògica de l'obtenció de dades, adaptació d'aquestes pel format de la llibreria i crida dels mètodes de la llibreria. La implementació d'un receptor en aquest punt del desenvolupament busca oferir una visió clara del funcionament d'OSNMA a la persona que l'executi i ajudar a perfilar els mètodes i variables necessaris que ha de contenir la llibreria. Aquesta fase és la més llarga i s'implementaran les diferents funcionalitats d'OSNMA de manera incremental per assegurar que cadascuna d'aquestes funciona correctament.

Finalment, es publicarà el paquet amb el nucli del protocol OSNMA al repositori de paquets oficial de Python PyPI. En tot moment es comentarà i documentarà el codi per futurs canvis o implementacions.

Capítol 4

Desenvolupament

En aquest capítol es seguirà el desenvolupament pràctic del projecte. Es dividirà en seccions que en la mesura del possible faran referència als punts esmentats a la metodologia.

4.1 Característiques bàsiques del projecte

4.1.1 Funcions criptogràfiques

El primer abans de desenvolupar el projecte era assegurar-se que es podrien realitzar les operacions criptogràfiques requerides pel protocol OSNMA.

Per realitzar les funcions de SHA-256, SHA3-224 i SHA3-256 es va optar per la llibreria estàndard de Python 3 `hashlib`. La decisió es va basar sobretot en que es un modul que ve per defecte a Python 3 y per tant es redueixen dependències i perquè, per aquest mateix motiu, esta optimitzat per la majoria de casos d'us. A més, esta perfectament documentat per aprendre'l a fer servir.

Per realitzar la funció d'HMAC-SHA-256 també es va triar una llibreria estàndard de Python 3, en aquest cas `hmac`. Aquesta llibreria ofereix les mateixes facilitats que `hashlib` tant en la seva importació com el seu ús.

Per realitzar la funció CMAC-AES es va triar la llibreria externa `pycryptodome` [Eij20] ja que proporcionava una interfície molt similar pel seu ús a la llibreria `hashlib` i a més consta d'una bona documentació per aprendre a fer-la servir.

Per realitzar les funcions de signatura digital per corba elíptica ECDSA P-224, P-256, P-384, P-521 es va optar per la llibreria externa `ecdsa` [War20] ja que esta ben documentada i ofereix informació del seu rendiment. Aquesta llibreria permet verificar signatures en menys de 0,005s,

molt per sota dels temps requerits en l'aplicació. A més, permet llegir les claus des d'un fitxer en format *pem*, fet que facilitarà els tests en un futur.

4.1.2 Representació dels valors dels camps

Per l'elecció del model de dades per la representació del valor dels camps primer hem de tenir en compte quins són els requeriments i dades amb les que es tractarà. Les dades dels camps del missatge es defineixen ocupant un nombre de bits concret i hem de poder concatenar-les per crear diferents missatges criptogràfics. A més, les funcions criptogràfiques actuen a nivell de byte, pel que hem de tenir en compte també aquesta conversió.

La primera aproximació fou fer servir el tipus de dades estàndard de Python `int`. Python ofereix mètodes estàtics integrats per convertir d'`int` a format `bytes`, binari o hexadecimal. El problema d'utilitzar el format `int` sorgia al intentar recuperar un valor guardat en la variable ja que els `ints` a Python no tenen mida determinada. Això vol dir que si guardem un valor que en el missatge original ocupa 8 bits, aquesta mida es perd i la nova mida és el bit 1 més significatiu. Per exemple, guardar en un `int` el valor '00110010' (50) al recuperar-lo després com a binari obtenim '110010'. Per tant, s'hauria de crear una classe nova que entengues la classe `int` i programar variables per guardar la mida i mètodes que retornessin el valor afegint '0' fins arribar a la mida. A més, per concatenar els valors `int` s'hauria de sobre-escriure el mètode de concatenació '+' per a que desplacés el valor `int` la mida del següent valor a concatenar i realitzés una operació `or` bit a bit amb aquest. Aquesta opció es va desestimar mentre es buscaven altres alternatives pel seu alt cost temporal, però queda oberta la possibilitat que sigui l'opció amb millor rendiment.

La segona opció plantejada va ser fer servir el tipus de dades estàndard a Python `ByteArray`. Aquest tipus de dades si que manté la mida original de les dades guardades i permet concatenar-se i obtenir la seva versió binaria o hexadecimal sense cap inconvenient. El problema es que funciona a nivell de byte i, per tant, si el valor carregat no encaixa en 8 bits, afegeix *padding* de 0s per davant i després al concatenar valors dona resultats erronis. En les dades dels camps OSNMA hi trobem valors que no són múltiples de 8 bits (sobretot molts per sota d'això) i per tant aquesta opció també es va descartar.

Una tercera opció era fer servir directament el format `String`, que guarda la mida, permet la conversió a `int` i hexadecimal i la concatenació és trivial. El problema sorgia al carregar les dades a les funcions criptogràfiques, ja que aquestes accepten la representació hexadecimal al que podem convertir la variable `String`. Però el format hexadecimal que accepten les funcions criptogràfiques és el que s'aconsegueix de la conversió de `ByteArray` o `bytes`. La diferencia rau en que un caràcter de la representació hexadecimal obtinguda d'`String` ocupa 8 bits a memòria, ja que es codifica en format ASCII, mentre que un caràcter hexadecimal real ocupa 4 bits a

memòria. Per tant tot i que es mostrin igual a l'usuari, internament no es pot utilitzar. A més la conversió d'`String` a `ByteArray` o `bytes` no es trivial.

Finalment, es va optar per una llibreria externa anomenada `bitstring` [Gri20] que conté una classe anomenada `BitArray`. Aquesta classe ofereix totes les característiques buscades de representació (`bytes`, binari, hexadecimal i *uint*) i a més permet carregar noves dades com a `String` binari, hexadecimal, octal, fitxers, `int`, etc. Tots aquests formats de carregar dades s'oferiran després des del paquet que implementa el protocol OSNMA cap enfora, permetent als receptors que l'implementin triar l'opció que millor els hi convingui.

4.2 Funcions bàsiques del protocol OSNMA

En aquesta secció es discutirà com s'han implementat les diferents funcions bàsiques així com algunes consideracions generals alhora de pensar-les que son transversals a elles. També es presentaran els vectors de test emprats.

4.2.1 Consideracions generals

4.2.1.1 Diccionari de camps i classe *Field*

Totes les funcions que implementarà la llibreria depenen en alguna manera dels camps llegits al missatge d'OSNMA i també han de crear un missatge per autenticar a partir d'aquest camps. Per tant, cal trobar una manera d'emmagatzemar aquests camps, no només les dades que contenen pel qual s'emprarà la classe `BitArray`. Per exemple, cal fer referencia d'alguna manera a quin valor guarda aquesta variable.

L'aproximació bàsica de crear tantes variables com camps hi hagi a OSNMA no es una manera pràctica d'organitzar el codi ja que es tenen variables amb la mateixa semàntica però no hi ha res al codi que ho indiqui. Es podrien agrupar en una llista, però normalment aquestes variables s'accediran de manera independent sense recórrer la llista, que es com realment s'aprofita aquesta estructura de dades. A més no hi ha cap relació entre la posició a la llista del camp i el camp que conté.

La primera opció real va ser crear un diccionari que agrupes tots els camps amb el nom de la variable com a clau i el valor del camp (en `BitArray`) com a valor. Aquesta opció es va fer servir inicialment en la primera implementació de les funcions, però de seguida es va canviar. El motiu va ser que hi ha més informació lligada al camp que no només les dades que conté, per exemple és útil que cada camp contingui el seu nom encara que estigui com a clau del diccionari i la seva descripció. A més, hi ha camps on el valor que contenen no representa només el valor, sinó que

te un significat. Per exemple el camp 'KS' que ocupa 4 bits (per tant pot tenir valors nominals del 0 al 15) indica la mida de la clau; però la mida de la clau no es directament el valor del camp sinó que es un valor entre 96 i 256 depenent del valor del camp. Era interessant mantenir aquesta informació lligada al camp i per això es va decidir crear un objecte que representes el camp.

Amb aquestes consideracions en ment es va decidir mantenir el diccionari però canviar el valor per un objecte que representes el camp OSNMA. Es va crear la classe `Field` que contenia com a atributs el valor del camp en un objecte `BitArray`, el nom del camp, i una funció que retorna la interpretació del valor del camp. L'us d'un atribut com a funció enlloc de codificar un mètode es que aquesta funció d'interpretació es diferent per cada camp. També es van afegir mètodes a la classe per recuperar i carregar el valor del camp en diferents formats (`BitArray`, `int`, hexadecimal).

En un primer moment es a començar a programar en aquesta organització, ja que en els test es disposava del valor del camp aïllat i es carregava directament a la variable del camp. Però quan es va començar a llegir els camps d'un flux de dades que representava tot el missatge es va fer palesa la necessitat d'afegir a la classe `Field` la mida del camp. Finalment, la classe `Field` va acabar tenint els atributs que indica la Figura 4.1 i tots els camps d'OSNMA van quedar agrupats en un diccionari.

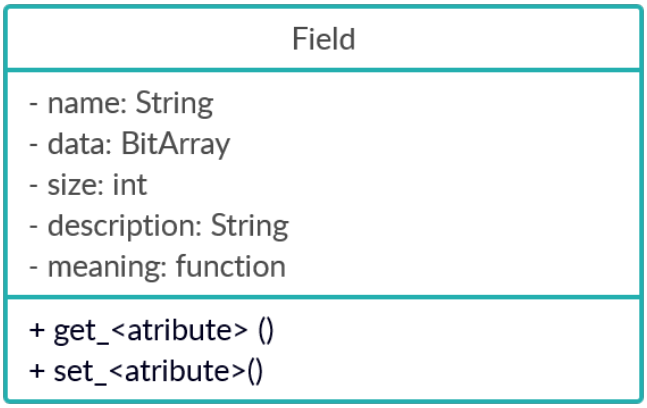


Figura 4.1: Estructura final de la classe `Field`.

4.2.1.2 Estructura dels missatges

Els camps d'OSNMA s'ordenen i concatenen de certa manera per crear els missatges d'autenticació. Per crear aquests missatges s'ha decidit utilitzar una estructura de llista Python que conté el nom del camp en cada posició. D'aquesta manera s'aprofita l'ordenació de l'estructura

```
message = BitArray()
for field in OSNMA_crypto['crypto_message_name']:
    message.append(OSNMA_data[field].get_data())
```

Figura 4.2: Codi exemple de la creació d'un missatge per autenticar.

```
raw_data = read_section_data()
bit_counter = 0
for field in OSNMA_sections['section_name']:
    if field == 'padding':
        load(field,raw_data[bit_counter:])
    else:
        load(field,raw_data[bit_counter:bit_counter+get_size(field)])
        bit_counter += get_size(field)
```

Figura 4.3: Codi exemple de la lectura d'un missatge d'una secció d'OSNMA arbitraria.

de llista que indica la posició del camp en el missatge. A més, el codi de la creació dels missatges es desacobla dels camps que contingui, ja que només modificant la llista emprada es modifica el missatge a crear. A més s'ha decidit agrupar totes les estructures dels missatges corresponents a la criptografia en un mateix diccionari. Això fa que la creació d'un missatge criptogràfic qualsevol sigui tan simple com es mostra a la Figura 4.2.

Alhora de llegir els camps del missatge HKROOT (tant si s'envia un DSM-KROOT com un DSM-PKR) també és útil tenir els camps ordenats en una llista, aixi simplement iterant per la llista i accedint als bits del missatge rebut es poden carregar les dades. En aquest punt es necessari que els camps continguin la seva mida, aixi es poden recórrer els bits de les dades del missatge simplement amb un comptador. A la Figura 4.3 es mostra l'algorisme simplificat de la lectura d'una de les seccions d'OSNMA (DSM-KROOT, DSM-PKR, NMA Header o DSM Header). La variable *raw_data* conte el valor en bits d'aquesta secció, la funcio `load` guarda els bits llegits a l'objecte *Field* corresponent i activa accions relacionades amb la lectura de certs camps. La comprovació del camp *padding* es necessaria ja que no tots els missatges encaixen exactament en un *subframe* d'I/NAV.

4.2.2 Verificació clau pública

Per autenticar una nova clau pública cal primer llegir el missatge DSM-PKR. La lectura es realitza com s'indica a la Figura 4.3. El missatge DSM-PKR indica la posició de la fulla que inclou aquesta clau l'arbre de Merkle amb un valor entre 0 i 15, ja que l'arbre de Merkle de l'especificació (Figura 4.4) té 16 fulles. El missatge també transmet els 4 nodes intermedis necessaris per verificar la fulla, el tipus de la nova clau (quina versió d'ECDSA es farà servir),

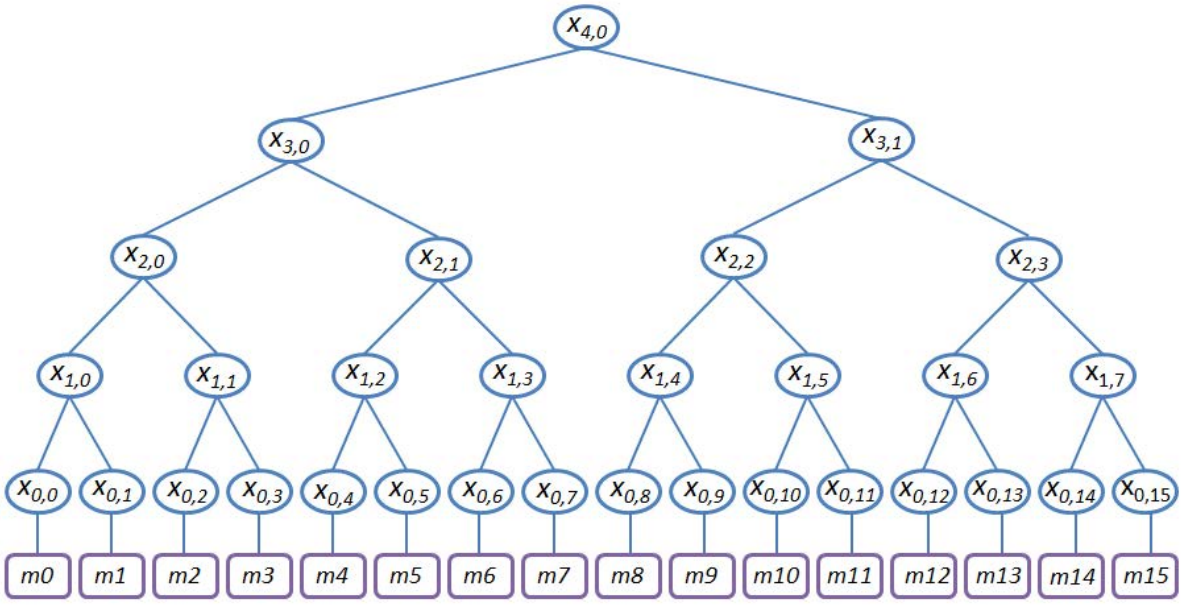


Figura 4.4: Forma de l'arbre de Merkle a [Fer19]

la ID de la clau a la que faran referencia les claus arrel de TESLA i la nova clau publica.

Per crear la fulla de l'arbre Merkle es concatenen els valors: tipus de clau pública, ID de la clau pública i la propia clau pública; i després es realitza un hash SHA256 d'aquest valor. La posició de la fulla dins l'arbre s'ha guardat a la variable m_{id} ens indicarà en quin ordre concatenar els nodes intermedis. Per trobar si el node actual es concatena per la dreta o per l'esquerra amb l'altre node del nivell podem aprofitar-nos de la posició del node actual m_{id}. Si es parell, l'altre node es concatena per darrere; si es parell, l'altre node es concatena per davant. Al pujar de nivell el número de nodes es divideix entre 2, per tant la variable m_{id} que ens indica la posició també es divideix entre 2 (de manera entera) i es torna a comprovar si es parell o imparell per concatenar el següent node.

El pseudo-codi per implementar aquesta operació es mostra en la Figura 4.5. La variable *mid* conté la posició de la fulla rebuda i la variable *itn_list* els nodes intermedis rebuts amb el missatge DSM-PKR. Un cop calculat el node arrel només cal comparar-lo amb el que es té guardat.

4.2.3 Verificació clau arrel

Per autenticar la clau arrel de la cadena TESLA (*KROOT*) es llegeix el missatge DSM-KROOT com s'indica a la Figura 4.3. Alguns d'aquests camps al ser carregats amb la funció `load` necessiten un tractament especial ja que impliquen variacions en el protocol. El camp *Key Size* indica la mida de les claus TESLA i es important que actualitzi de manera síncrona la mida del

```
node = sha256(message)
for itnode in itn_list:
    if mid%2 == 0:
        node = sha256(node + itnode)
    else:
        node = sha256(itnode + node)
    mid = mid//2
```

Figura 4.5: Codi exemple de la construcció del node arrel d'un arbre de Merkle

camp que conté la clau arrel abans de seguir llegint el missatge DSM-KROOT. En cas contrari no sabríem la mida en bits de la clau quan toques llegir-la del missatge. També es important el tractament dels camps *Hash Function* i *MAC Function* que definiran quines funcions es faran servir.

Per crear el missatge a autenticar el segueix l'algorisme descrit a la Figura 4.2. En aquest missatge d'autentificació no només s'utilitzen els camps enviats al missatge DSM-KROOT, sinó que també es concatena el valor del camp NMA Header que es manté constant en tot el missatge DSM.

Per autenticar el missatge s'utilitzen les funcions del paquet `ecdsa` descrit a la secció 4.1.1 i la clau pública es suposa guardada en un fitxer en el format estàndard *pem*. El pseudocodi de la implementació es pot veure a la Figura 4.6. Primer s'obté el descriptor de fitxer del fitxer *pem* amb la clau pública fent servir al funció `with` per bona praxis i evitar deixar el fitxer obert i bloquejat en cas d'error. A continuació s'instancia l'objecte de verificació del paquet `ecdsa` dins d'una sentencia `try/excep` per detectar possibles errors en la lectura. Finalment, es crida a la funció `verify` de l'objecte que conté la clau pública i se li passa com a paràmetre la signatura digital llegida del missatge DSM-HKROOT i el missatge a autenticar creat a partir dels camps.

Internament la funció `verify` desencriptarà la signatura digital rebuda fent servir al clau pública i el compararà amb el missatge a autenticar. Si els missatges són idèntics, la clau arrel K_0 que forma part d'aquest missatge queda autenticada, ja que només el propietari de la clau privada ha pogut crear aquesta signatura. I sabem que la clau pública pertany a la clau privada del sistema Galileo perquè s'ha autenticat amb l'arbre de Merkle.

4.2.4 Verificació clau TESLA

Per autenticar una clau TESLA cal aplicar la funció resum (*hash*) a la clau n vegades i comparar-la amb la clau arrel K_0. La funció resum s'aplica a la concatenació de la clau TESLA, el GST (Galileo Satellite Time) del *subframe* on s'ha rebut la següent clau a generar i un valor alfa constant a tota la cadena TESLA i obtingut al missatge DSM-KROOT.

```
with open(pub_key) as f:
    try:
        vk = ecdsa.VerifyingKey.from_pem(f.read(), hashfunc=hash_function)
    except IOError as e:
        print("I/O error({0}): {1}".format(e.errno, e.strerror))

try:
    verification_result = vk.verify(read_digital_signature,
                                    crafted_authentication_message)
except ecdsa.BadSignatureError as e:
    verification_result = False
finally:
    return verification_result
```

Figura 4.6: Codi exemple de l'autenticació de la clau arrel K_0 amb la clau pública guardara en un fitxer *pem*.

La primera consideració és quants cops cal aplicar la funció resum; o en altres paraules, quina és l'index de la clau dins la cadena TESLA. Per trobar l'index de la clau actual s'ha utilitzat la següent formula:

$$index = \frac{GST_{SF} - GST_0}{30} \cdot NS \cdot NMACK + NS \cdot Pos + SVID$$

On GST_{SF} és el temps del *subframe* on s'ha rebut la cadena actual, GST_0 es el temps en que ha començat a aplicar-se la cadena (el temps rebut al DSM-KROOT amb la clau arrel), NS el número de satèl·lits Galileo, $NMACK$ el nombre de blocs MACK dins del missatge MACK (rebut al DSM-KROOT), Pos es la posició del bloc de la clau dins dels $NMACK$ blocs, i $SVID$ (Space Vehicle ID) l'identificador del satèl·lit que esta rebent les dades.

El primer terme de l'equació calcula el nombre de *subframes* que ha passat des de l'inici de la cadena $\frac{GST_{SF}-GST_0}{30}$ i multiplica el nombre de *subframes* pel número de satèl·lits de Galileo (ja que cada satèl·lit transmet una clau diferent) i pel nombre de claus que s'envien per *subframe* ($NMACK$). A continuació, es multiplica la posició de la clau dins d'aquest *subframe* pel número de claus que s'envien a cada posició (que és el nombre de satèl·lits). Finalment, es suma l'identificador del satèl·lit actual, ja que les claus TESLA s'ordenen en funció de l'$SVID$.

Un cop s'ha calculat l'index de la clau que es vol verificar, es pot començar a aplicar la funció resum. Però com ja hem vist, per aplicar-la cal el GST_{SF} de la següent clau que es vol calcular. Per tant, s'aplica la següent formula a cada índex per trobar el GST_{SF} que s'aplica en aquesta funció:

$$GST_{SF} = GST_0 + 30 \cdot floor(\frac{m-1}{NS \cdot NMACK})$$

Aquesta formula és més senzilla ja que no s'ha de tenir en compte l'*offset* de claus dins de cada *subframe*, fet que es corregeix amb la funció $floor$. Per calcular la clau arrel K_0 el temps GST_{SF} és $GST_0 - 30$ per definició. Amb aquestes dues funcions es pot anar iterant per la cadena de claus fins arribar a la clau arrel.

Un cop calculada la clau arrel, es compara amb la K_0 guardada i verificar que pertany a la cadena. El problema sorgeix que quan la cadena es molt llarga el cost computacional d'arribar a K_0 augmenta i que si una clau ja s'ha calculat que pertany a la cadena, si es torna a obtenir aquest valor ja es pot determinar que la clau que s'esta verificant també pertany a la cadena. Utilitzar altres claus de la cadena ja verificades per verificar noves claus s'anomena utilitzar claus flotants o *floating keys*.

Per implementar les claus flotants la primera idea va ser guardar el valor de la clau directament en un diccionari per aquella cadena TESLA accessible per l'index de la clau. Però guardar només el valor de la clau obligava a re-calcular el valor de GST_{SF} cada cop que es volgués realitzar algun càlcul extra. A més, pot ser interessant guardar la cadena en opcions de *logging* i la informació extra és útil. Per això es va decidir crear la classe `KeyEntry` (Figura 4.7), per guardar tota la informació lògica relacionada amb una clau TESLA.

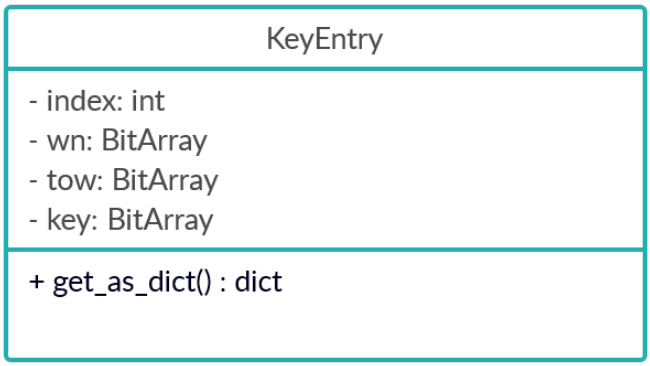


Figura 4.7: Estructura de la classe `KeyEntry`.

Per tant el diccionari guarda objectes *KeyEntry* con a valor i l'index de la clau com a clau del diccionari. També es genera un diccionari temporal que guarda totes les claus que es recorren quan s'esta verificant una clau i, quan la clau s'ha verificat, el diccionari temporal s'uneix al diccionari que es té guardat amb totes les claus anteriors. D'aquesta manera no s'embruta el diccionari amb claus que potser son falses.

4.2.5 Verificació MAC

El missatge d'OSNMA MACK esta dividit en $NMACK$ blocs (entre 1 i 3). Cada bloc MACK conté una serie d'entrades MAC i la clau TESLA que les autentica. Totes aquestes entrades MAC menys la primera contenen el valor de la MAC, la SVID del satel·lit d'on autentiquen les dades, un camp anomenat ADKD (Authetication Data & Key Delay) que descriu quina part del missatge autentica i el camp IOD (Issue Of Data) amb un significat diferent depenent del ADKD i que complementa com autenticar les dades.

La primera entrada MAC del primer bloc MACK s'anomena MAC0 i sempre autentica les dades del propi satèl·lit en l'actual *subframe* corresponent amb un valor ADKD = 0. A més del valor MAC0, la primera entrada del primer bloc conté un camp anomenat MACSEQ, que autentica les entrades MAC no predeterminades del primer bloc MACK.

El ADKD i de quin satèl·lit autentiquen les dades cada entrada MAC venen predeterminades per una taula pública. El valor de MACLT (MAC Lookup Table) llegit al DSM-KROOT indica quina entrada de la taula correspondrà amb les MAC que es llegiran, d'aquesta manera s'eviten intents d'introduir noves dades. Però hi ha certes entrades al primer bloc de MACK que s'anomenen flexibles i no estan predeterminades, i es el camp MACSEQ de la primera entrada del primer bloc el que les autentica.

En la primera implementació de la llibreria es donarà suport només a la verificació dels camps MAC0 i MACSEQ, però s'han plantejat i preparat les estructures de dades per a la futura implementació.

Per verificar el camp MAC0 es concatenen els camps de la mateixa forma que a la Figura 4.2. Però el camp *navdata* s'ha de construir a part d'acord amb la definició d'ADKD = 0. Per gestionar totes les mascares possibles segons el valor d'ADKD s'ha creat una llista on cada element correspon a una mascara. La mascara esta formada per una llista de diccionaris anònims que contenen cadascun la pagina del missatge de navegació I/NAV al que fan referencia i una llista amb tuples dels bits consecutius que s'han de llegir d'aquesta pagina. Cada tupla indica el bit inici i bit final de la seqüencia de bits a llegir. La Figura 4.8 mostra com s'organitza aquesta informació. El valor de *word* es purament informatiu, però en el futur es podria fer servir per filtrar directament en busca d'aquesta *word* al missatge I/NAV. Al implementar-se en diccionaris l'addició en el futur d'altres ADKD es presenta senzilla.

Aquests valors es concatenen en l'ordre de la llista que correspon a la ADKD = 0 i aixi es crea el camp *navdata* per l'autenticació, al que s'aplicarà a funció MAC definida al DSM-KROOT per aquesta cadena i compararà amb el camp MAC0.

Pera autenticar el camp MACSEQ la operació és més senzilla ja que només cal concatenar la ID del satèl·lit actual, el GST_{SF} i els camps flexibles que indiqui l'entrada a la taula MACLT

```
adkd_masks = [
    [
        {
            'word': 1,
            'page': 10,
            'bits': [(8,114),(122,136)]
        },
        {
            'word': 2,
            'page': 0,
            'bits': [(8,114),(122,136)]
        },
        {
            'word': 3,
            'page': 11,
            'bits': [(8,114),(122,138)]
        },
        {
            'word': 4,
            'page': 1,
            'bits': [(8,114),(122,136)]
        },
        {
            'word': 5,
            'page': 12,
            'bits': [(8,75)]
        }
    ]
]
```

Figura 4.8: Mascarà ADKD d'exemple per ADKD = 0.

(*MAC Lookup Table*). La MACLT s'ha codificat com una llista on cada entrada es un diccionari anònim que conté la informació de l'entrada a la taula (Figura 4.9). Per la creació del missatge MACSEQ, s'itera sobre el primer MACK bloc de la llista `sequence` del diccionari que representa l'entrada a la taula MACLT buscant els camps FLX i es concatenen.

4.2.6 Test de les funcions

Per assegurar el correcte funcionament de cadascuna de les funcions s'han dissenyat una serie de tests unitaris. Aquests test utilitzen dades proporcionades a [FH16b] de les quals es coneix el resultat esperat. S'han dissenyat tests per les funcions de verificació de la clau publica, verificació de la clau arrel K_0, verificació de la clau TESLA, verificació de la MAC i la MACSEQ.

Els test s'han anat adaptant al desenvolupament del projecte, és a dir, al principi avaluaven la implementació de les funcions en un fitxer a part i a mida que s'integraven dins la classe del paquet que les acabaria contenint finalment s'anaven modificant. Cada cop que es realitzava un canvi substancial es tornaven a executar per assegurar que el desenvolupament de nous mòduls no introduïa errors.

4.3 Creació del paquet: `osnma_core`

En aquesta secció es discutirà la integració de les funcions bàsiques d'OSNMA dins de la classe OSNMACore i l'organització d'altres fitxer auxiliars que conformaran el paquet Python `osnma_core`. També s'explicarà com s'han gestionat els errors seguint el llibre d'estil de Python i com s'ha documentat i publicat el paquet.

4.3.1 Gestió de les dades i integració de les funcions

Per gestionar el model de dades dins del paquet `osnma_core` s'ha decidit crear una carpeta que contingui els fitxers amb les dades comuns, per tal de desacoblar el codi i que sigui més fàcil de llegir i modificar. Aquests fitxers contenen la llista de camps d'OSNMA, l'estructura dels missatges rebuts d'OSNMA i l'estructura dels missatges criptogràfics a crear, les mascares ADKD i la taula MACLT.

Per agrupar les funcions i carregar dades s'ha creat la classe OSNMACore. Aquesta classe implementa totes les funcions discutides a 4.2 com a mètodes de la classe. Les funcions utilitzaran les dades dels camps OSNMA guardades en el diccionari comú de la classe i també oferirà funcions de carrega de noves dades.

Per carregar noves dades als camps d'OSNMA de la classe s'ha creat el mètode `load`. Aquesta

```
mac_lookup_table = [
    {
        'ID': 0,
        'sections': 1,
        'NMACK': 1,
        'MACs': 14,
        'sequence': ['00S','00E','00E','00E','00E','00E','00E','00S','00E','00E'
                                                ,'00E','00E','00E','00E']
    },
    {
        'ID': 1,
        'sections': 1,
        'NMACK': 1,
        'MACs': 14,
        'sequence': ['00S','00G','00G','00G','00G','00E','00E','00E','00E','00E'
                                                ,'00E','00S','00G','00G']
    },
    ...
    {
        'ID': 10,
        'sections': 1,
        'NMACK': 2,
        'MACs': 5,
        'sequence': [['00S','FLX','FLX','FLX','FLX'],['00S','00E','00E','00E','
                                                00E']]
    },
    {
        'ID': 11,
        'sections': 1,
        'NMACK': 2,
        'MACs': 5,
        'sequence': [['00S','FLX','FLX','00E','11S'],['00S','00E','00E','00E','
                                                12S']]
    },
```

Figura 4.9: Fragment de la codificació de la *MAC Lookup Table*.

funció rep com a paràmetre el nom del camp i les dades corresponents a aquell camp, accedeix al diccionari de camps i guarda les dades. A més, i motiu de pel que ha d'existir aquest mètode, realitza accions addicionals depenent del camps que es carrega com ara modificar la mida de certs camps o indicar l'us d'una funció criptogràfica o una altra. També existeix el mètode `load_batch` que permet carregar varis camps alhora rebent les dades com a un diccionari. Internament, segueix cridant al mètode `load` per cada una de les entrades.

La classe ofereix els mètodes de verificació dels diferents components d'OSNMA per ser cridats directament des del receptor on s'implementi i s'executaran fent servir els camps OSNMA carregats prèviament a la classe. Però també ofereix l'opció de tractar directament la seqüencia de bits corresponent al missatge de DSM-HKROOT, DSM-PKR i MACK. Si es tria aquesta alternativa, la classe s'encarregarà de fragmentar el missatge i carregar els camps al diccionari intern per després cridar als metodes pertinents de verificació.

Aquestes dues alternatives es proporcionen per acomodar diferents implementacions de receptors. Si el receptor no coneix la lògica del protocol OSNMA pot simplement agrupar els bits del camp i deixar que la classe s'encarregui. En canvi, si el receptor vol saber en tot moment el que esta llegint i fent, pot aprofitar-se de les estructures proporcionades per la classe i realitzar ell mateix la lectura dels camps i després cridar a les funcions de verificació pertinents.

La classe OSNMACore també actua com a interfície per recuperar els camps OSNMA, permetent obtenir les dades (en diferents formats), la mida, el significat o la descripció de tots els camps ONSMA guardats al diccionari intern. També permet retornar el diccionari que conte totes les claus TESLA verificades de la cadena actual.

Tots els mètodes que accepten dades d'entrada que corresponen amb seqüencies de bits accepten els següents formats:

- Objecte `BitArray` de la llibreria `bitstring`.
- Objecte Bytes o `ByteArray` de la llibreria estàndard de Python.
- `String` binari: Indicat amb els valors inicials '0b'.
- `String` hexadecimal: Indicat amb els valors inicials '0x'.
- `String` Octal: Indicat amb els valors inicials '0o'.

D'aquesta manera s'ofereix suport a diferents implementacions del receptor sense que hagi de realitzar transformacions de dades excessives. En qualsevol cas, les dades d'entrada son convertides a objectes `BitArray` abans de procedir amb les operacions, ja que es el format de dades comú a tot el paquet.

4.3.2 Gestió d'errors: excepcions

Al ser un paquet extern que faran servir diferents desenvolupadors, la gestió dels errors que es produeixin (tant pel seu mal ús com per errors interns propis) té una importància cabdal. La manera estàndard de tractar errors es mitjançant excepcions i blocs `try/excep`. En el cas d'aquest paquet, s'han intentat llençar excepcions sempre que s'intentin cridar a mètodes que no disposen de totes les dades per ser executats. Per exemple, cridar el mètode per verificar una MAC sense haver definit la funció MAC a utilitzar (ja sigui pel camp d'OSNMA *MF* o manualment).

Per aquest projecte s'han creat dues excepcions pròpies per gestionar errors. La primera anomenada `MissingFieldData` per llençar quan s'esta construint un missatge i un camp no te dades. L'altra, `MissingFieldSize`, per indicar que un camp no te mida definida i no es poden llegir dels bits del missatge.

També s'han intentat cobrir totes les operacions que depenen de llibreries externes com ara la creació de dades `BitArray`, la lectura de fitxers i autenticació amb blocs `try/excep`. D'aquesta manera s'aïllen els erros i s'ofereix una informació clara i concisa al desenvolupador del que esta succeint i si esta fent operació cosa errònia.

4.3.3 Documentació i publicació del paquet

Per documentar el paquet s'han escrit comentaris en el format *reStructuredText* [Goo16] indicant que fa el paquet, el funcionament de cada classe i el funcionament de cada funció. Aquest format de text permet definir per cada funció que fan els atributs i de quin tipus de dades han de ser. També permet definir quin valor retornarà la funció. Tota aquesta informació la interpreten la majoria d'IDEs i la mostren al programador que estigui fent servir la nostra llibreria i això és el que busquem. A més, és una manera ordenada de documentar que també ajuda a mantenir el paquet i actualitzar-lo en el futur.

Els comentaris en format *reStructuredText* també es poden llegir des de programari extern i generar fitxers HTML o pdf amb la documentació del projecte. En aquest cas s'ha utilitzat el programari *Sphinx* [Bra07] per programar una petita web que mostri la documentació. També s'ha programat amb *Sphinx* la impressió en pdf de la documentació.

Per distribuir el paquet de manera oficial pel repositori *Python Package Index* (PyPI) i que per tant es pugui instal·lar en qualsevol distribució de Python 3 mitjançant la comanda *pip install* s'ha hagut de complir amb la normativa del repositori. Aquesta normativa obliga a definir una llicencia pel codi per la qual s'ha triat GPLv3 [Fre07] ja que obliga a tot el codi derivat a seguir sent obert i no privatiu. També s'han hagut de crear fitxers de configuració indicant el nom del paquet (`osnma_core`), el creador, la versió i les dependències necessàries que

s'instal·laran alhora que el paquet.

4.4 Creació del receptor: OSNMA_receiver

En aquesta secció es detallaran les decisions de disseny del receptor d'OSNMA que s'ha decidit programar a partir de la llibreria OSNMACore. La classe del receptor s'anomena OSNMA-Receiver i està al fitxer *osnma_receiver.py*. Aquest receptor s'ha decidit que actuï de manera divulgativa mostrant les operacions que realitza el protocol, i no tant com un receptor real. Així el receptor pot servir com a test de les funcions del paquet `osnma_core` que utilitza i per ensenyar el funcionament del protocol a terceres persones.

4.4.1 Lectura de les dades

Les dades que s'utilitzaran pel receptor estan generades per un simulador del missatge I/NAV i representen dades reals. Es disposa un fitxer *csv* en el que cada entrada conté una pàgina del *subframe* I/NAV, junt amb la SVID del satèl·lit que l'envia, el WN i TOW d'emissió i les dades que conté.

Per tant el primer que ha de fer el receptor és anar llegint les dades des del *csv*. A més, la implementació actual del receptor no suporta autenticació creuada pel que s'han de filtrar les dades per aquelles de satel·lit amb SVID igual a 1.

Per tant, el receptor va llegint les dades del fitxer *csv* per l'ordre de WN i TOW i les processa. De cada pàgina llegida, s'ha d'accedir al camp HKROOT que correspon als bits entre la posició 138 i 146 (8 bits). També s'ha d'accedir als bits entre la posició 146 i 178 (32) que formen el missatge MACK.

4.4.2 Lògica del receptor: lectura dels camps OSNMA

El primer del que ha de disposar el receptor és d'una manera de controlar quan acaba un *subframe* i comença un altre, ja que implicarà la transmissió d'un nou HKROOT bloc i MACK bloc. Per tant, detectar l'inici d'un nou *subframe* és necessari per reiniciar certes variables d'estat del receptor. A més, l'operació de verificació de la clau TESLA necessita el temps GST del *subframe* i la funció que controla la pàgina del *subframe* és el lloc ideal per gestionar aquestes variables. Totes aquestes accions les realitza la funció `process_subframe_page` que es crida amb cada lectura del fitxer *csv*.

Un cop gestionada la pàgina del *subframe*, el tractament és diferent pels 8 bits del HKROOT llegits i pels 32 del MACK.

4.4.2.1 HKROOT

La primera pàgina del *subframe* sempre conté el NMA Header. La segona sempre conté el DSM Header i les restants formen part del DSM bloc, que pot ser KROOT o PKR segons indiqui el DSM Header. Per tant hi ha una estructura `if/else` que avalua la variable que indica quina pàgina s'esta llegint del *subframe* i en funció d'aquesta crida a la funció de gestió que pertoqui. Per triar si crida a la funció que gestió de KROOT o PKR per les pàgines més grans que 2 es basa en el valor de DSM-ID indicat al DSM Header que acaba de llegir.

El mètode que llegeix el NMA Header simplement el llegeix i carrega a l'objecte OSNMACore instanciat. Si el valor que llegeix es diferent al que te guardat, l'envia a la funció per escriure'l per pantalla.

El mètode que llegeix el DSM Header, a part de llegir-lo i guardar-lo a l'objecte OSNMACore, també ha de gestionar quan acaba un missatge DSM i comença un altre. El DSM Header indica la ID del missatge DSM (per poder-lo reconstuir a partir dels blocs) i el número del bloc DSM. El primer camp dels dos possibles missatges DSM (KROOT i PKR) indica quants blocs ocupa el missatge. Per tant, el mètode ha de comprovar si ja s'ha ha acabat de llegit l'ultim missatge DSM i aquest n'és un de nou.

Es necessari saber si el *subframe* actual conté l'últim bloc del missatge DSM per realitzar les operacions de verificació pertinents quan s'acabi de llegir. També es necessari saber si es l'inici d'un nou missatge DSM per inicialitzar variables relacionades amb la lectura dels camps.

El mètode que llegeix el camp DMS-KROOT podria simplement concatenar tots els valors de 8 bits de cada pàgina de cada *subframe* fins que s'hagi llegit tot el missatge DMS-KROOT i llavors cridar a la funció de la llibreria OSNMA que processa tot el missatge DMS-KROOT. Però com el receptor ha de mostrar el funcionament del protocol de manera divulgativa s'ha optat per, a mida que llegeix el camp del 8 bits, el vagi carregant a l'objecte OSMNACore i mostrant-ho per pantalla. Però la llibreria `osnma_core` també permet realitzar la carrega de manera manual i a més proporciona les estructures de dades del missatge per a facilitar la feina al receptor.

El que s'ha de tenir en compte alhora de llegir els camps a partir de blocs de 8 bits de cada pàgina són els possibles casos que es poden donar. Primerament, es defineixen 3 comptadors: el primer (`bit_count`) indica el número de bits llegits del bloc de 8, el segon (`msg_pos`) indica quin camp del missatge s'esta llegint i el tercer (`field_bits`) indica quants bits del camp s'han llegit. Amb aquests comptadors definits ens podem trobar en 4 situacions alhora de llegir un camp:

- El camp a llegir cap en els 8 bits: Aquest cas és el més simple, es llegeix el camp, es carrega a l'objecte OSMNACore i s'avança el comptador `msg_pos` en 1 camp i el comptador

`bit_count` en la mida del camp llegit. Si el valor de `bit_count` és inferior a 8 es procedeix amb el següent camp, sinó s'espera a la següent pàgina.

- El camp a llegir no cap en els 8 bits: Es guarda la part del camp dels 8 bits en una variable temporal, s'incrementa el comptador `field_bits` en els bits llegits i s'espera a la següent pàgina.
- S'esta a mitja lectura i la part restant no cap en els 8 bits: Es concatenen els 8 bits llegits a la variable temporal que conté les parts del camp i s'incrementa el comptador `field_bits` en 8.
- S'esta a mitja lectura i la part restant acaba en els 8 bits: Es concatenen els bits restants a la variable temporal i es guarda la variable que conté el camp a l'objecte OSNMACore. S'avança el comptador `msg_pos` en 1, s'incrementa el comptador `bit_count` en els bits llegits i es posa a 0 `field_bits`. Es llegeix el següent camp si `bit_count` és inferior a 8 bits.

El missatge DMS-PKR no s'ha considerat en aquest receptor ja que les dades de les que es disposa son d'un cas nominal sense canvi de clau pública, però la lògica seria la mateixa i esta perfectament suportat al paquet `osnma_core`.

4.4.2.2 MACK

La lectura dels 32 bits per pàgina del camp MACK és més senzilla ja que és independent per cada *subframe*. El procediment que s'ha dissenyat al receptor consisteix en concatenar els 32 bits de cada pàgina del *subframe* fins a crear el missatge MACK d'aquell *subframe*. Si la clau arrel (K_0) encara no s'ha verificat, es guarda el valor del missatge en una llista junt amb el GST_{SF} del *subframe* i es procedeix amb el MACK del següent *subframe*.

4.4.3 Lògica del receptor: verificació dels missatges

La verificació del la clau arrel (K_0) es duu a terme quan el receptor detecta que ha acabat de llegir un missatge DMS que en que la ID pertany a un missatge DMS-KROOT. La verificació la computa l'objecte de la classe OSNMACore i el receptor es guarda el resultat.

Si el resultat de la verificació es afirmatiu, el receptor crida a l'objecte OSNMACore per la verificació de les claus TESLA i les MACs del missatge MACK. Si hi ha missatges MACK pertanyents a la cadena esperant a la verificació de la clau arrel, aquests es verifiquen primer i després l'actual. En els següents *subframes* rebuts, com la clau arrel de la cadena ja està autenticada, es verificarà sense espera el valor del missatge MACK. L'objecte OSNMACore

conté tota la informació necessària per tractar el missatge MACK ja que l'ha rebut al missatge DMS-KROOT i el receptor simplement ha de passar-li els bits del missatge.

4.4.4 Opcions de *logging*

El receptor pot mostrar per pantalla la informació que va llegint, les operacions que realitza i el seu estat. Per disminuir o augmentar la informació que es mostra a l'usuari s'han implementat certes variables que regulen quins paràmetres es mostren durant l'execució. Aquestes funcions s'han implementat fent que quan el receptor vol escriure per pantalla, en lloc de cridar a la funció estàndard `print`, crida a una funció pròpia a la que es passa el text a mostrar i la variable de *logging* que correspon. En funció del valor de la variable definit per l'usuari, el text es mostrarà o no. La mateixa aproximació es pot aplicar en el futur per guardar *logs* en fitxers.

Aquestes variables es defineixen quan s'instancia la classe del receptor OSNMA_receiver i són:

- `verbose_mack`: Si la variable és `True`, el receptor mostrarà per pantalla cada cop que verifiqui una MAC o MACSEQ. Si es `False` només mostrarà quan falli la verificació.
- `verbose_headers`: Si la variable és `True`, sempre es mostrara per pantalla el NMA Header i DSM Header encara que continguin els mateixos valors que el *subframe* anterior.
- `only_headers`: Si la variable és `True`, no es processa el missatge DSM ni MACK. Pensada per realitzar execucions llargues en busca d'errors.

Independentment de la configuració de les variables, el receptor mostrarà per pantalla el WN i TOW cada cop que es llegeixi un *subframe*. També esta programat per guardar en el *path* que se li passa com a paràmetre un fitxer *csv* amb les claus de la cadena que s'han validat. Finalment, per no saturar la consola, s'ha definit la variable `max_iter` que defineix quantes pàgines I/NAV es llegiran del fitxer.

Capítol 5

Resultats

El paquet *osnma_core* que implementa tots els objectes, variables i funcions per gestionar el protocol OSNMA es pot trobar al repositori oficial PyPI en el següent enllaç:

`https://pypi.org/project/osnma-core/`

La instal·lació es pot realitzar amb la comanda `pip install osnma-core` o descarregant el paquet des de la web i afegint-lo a la variable global `PYTHONPATH`. El codi del paquet es pot consultar al GitHub:

`https://github.com/Algafix/osnma_core`

I la documentació del paquet es pot trobar en format web a l'enllaç:

`https://osnma-core.readthedocs.io`

El receptor OSNMA junt amb les dades reals del missatge I/NAV utilitzades per la simulació i un escenari de mostra del seu funcionament nominal es troba allotjat al GitHub:

`https://github.com/Algafix/osnma-receiver`

Per executar en local l'escenari de simulació nominal cal primer clonar o descarregar el Git i després realitzar els següents passos:

```
$ cd <directori del git>
$ pip install requirements.txt
$ python3 scenario1.py
```

També es pot obrir el fitxer *scenario1.py* i modificar el valor de les variables `verbose_mack`, `verbose_headers`, `only_headers` i `max_iter` per aconseguir el comportament desitjat. La Figura 5.1 mostra el comportament estàndard amb la configuració `verbose_mack = True` i les altres variables a `False`.

A la subfigura 5.1.A el receptor mostra que ha començat a processar el *subframe* corresponent al TOW: 432180 i com va llegint els camps de 8 bits del HKROOT de cada pàgina I/NAV de les 15 que conformen el *subframe*. El següent *subframe* comença al TOW: 432210, el NMA Header es manté igual, en canvi el DSM Header varia ja que el valor DSM Block indica que aquest es el segon bloc del missatge; i el receptor s'encarrega de mostrar-ho. Llavors continua la lectura per on l'ha deixat després del primer *subframe*, començant a llegir al clau arrel.

La subfigura 5.1.B mostra com després de 7 *subframes* s'esta acabant de llegir la signatura digital de la clau arrel. Llavors la clau es verifica i, com el resultat es afirmatiu, es comencen a validar els camps MACK que estaven a l'espera. El receptor mostra que el primer camp MACK que es valida es el primer que s'havia rebut al *subframe* TOW: 432180 junt amb l'inici del missatge KROOT a la subfigura anterior.

Finalment, a la subfigura 5.1.C s'acaben de verificar les MACK pendents i es verifica la corresponent al *subframe actual*. Llavors es llegeix el següent *subframe* i com la cadena de claus es la mateix el missatge DMS-KROOT és el mateix que l'anterior. El receptor ho detecta ja que el camp ID del DMS Header és el mateix, així que per estalviar recursos no processa el missatge DMS i s'espera a l'inici del següent. El que si que verifica es el camp MACK.

```
(tfg-telecos) algafix@algafix-X555LJ:~/tfg-telecos/OSNMA_receiver$  cd /h
ome/algafix/tfg-telecos/OSNMA_receiver ; env /home/algafix/.virtualenvs/t
fg-telecos/bin/python /home/algafix/.vscode/extensions/ms-python.python-2
020.8.101144/pythonFiles/lib/python/debugpy/launcher 35273 -- /home/algaf
ix/tfg-telecos/OSNMA_receiver/scenario1.py

New Subframe:
        WN: 1018 TOW: 432180
Read NMA Status: Operational (0b10)
Read Chain ID: 0 (0b00)
Read Chain and Public Key Status: Nominal (0b001)
Read DSM ID: DMS-KROOT ID (0b0000)
Read DSM Block ID: 1 (0b0000)

Start DSM Message

Read Nb. of Blocks: 7 (0b0001)
Read Public Key ID: 0 (0b0000)
Read Chain ID of KROOT: 0 (0b00)
Read Nb. of MACK blocks: 240 (0b10)
Read Hash Function: SHA256 (0b00)
Read MAC Function: HMAC-SHA-256 (0b00)
Read Key Size: 112 (0b0010)
Read MAC Size: 16 (0b0011)
Read MAC Lookup Table: 8 (0b00001000)
Read Reserved: 0 (0b00)
Read MACK Offset: No Offset (0b00)
Reading KROOT Week Number (4/12)
Read KROOT Week Number: 1018 (0b001111111010)
Read KROOT Time of Week (hours): 120 (0b01111000)
Reading alpha (8/48)
Reading alpha (16/48)
Reading alpha (24/48)
Reading alpha (32/48)
Reading alpha (40/48)
Read alpha: fbca34569875

New Subframe:
        WN: 1018 TOW: 432210
Read DSM Block ID: 2 (0b0001)
Reading Key Root (8/112)
Reading Key Root (16/112)
```

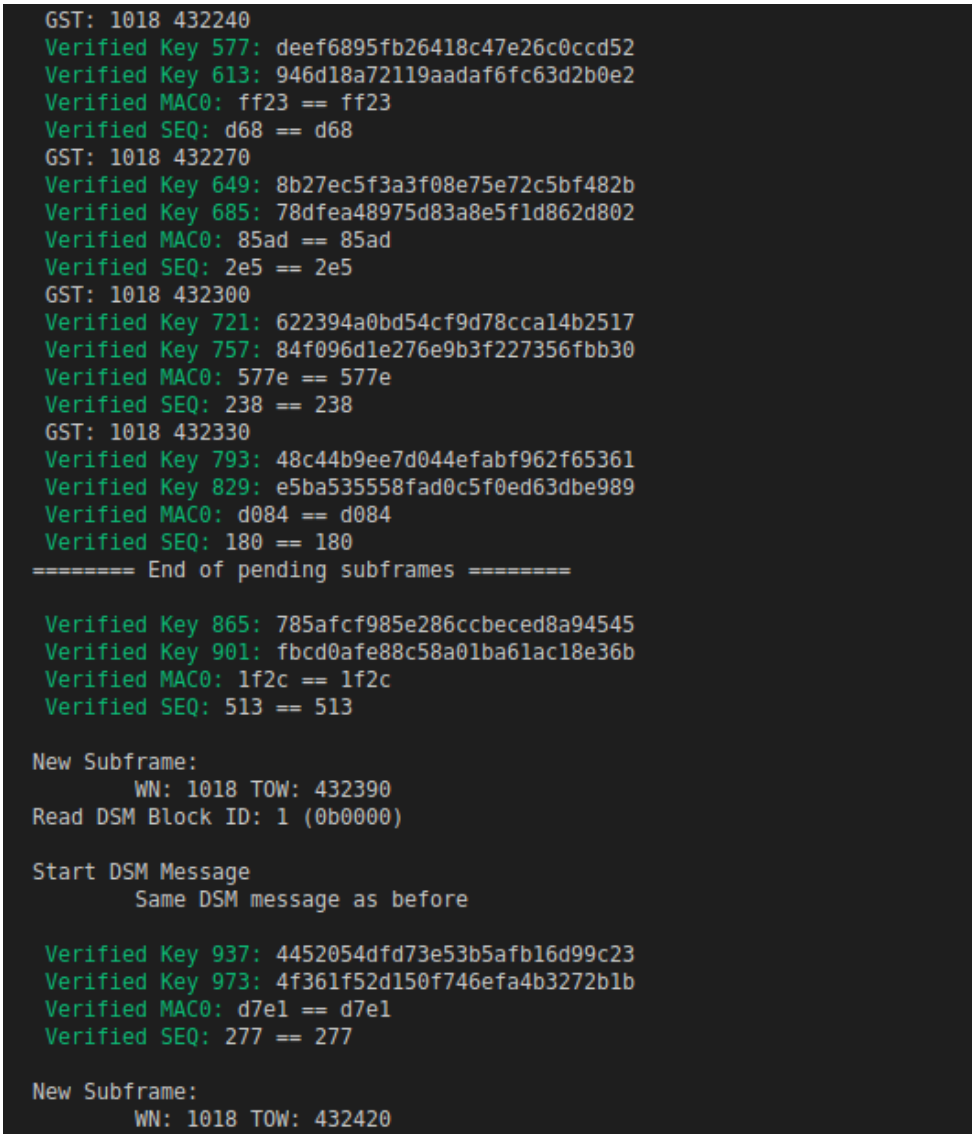
5.1.A Primera part de l'execució: lectura dels camps del primer *subframe* DMS-KROOT i inici de la lectura de la clau arrel al segon *subframe*.

```
New Subframe:
        WN: 1018 TOW: 432360
Read DSM Block ID: 7 (0b0110)
Reading Digital Signature (416/512)
Reading Digital Signature (424/512)
Reading Digital Signature (432/512)
Reading Digital Signature (440/512)
Reading Digital Signature (448/512)
Reading Digital Signature (456/512)
Reading Digital Signature (464/512)
Reading Digital Signature (472/512)
Reading Digital Signature (480/512)
Reading Digital Signature (488/512)
Reading Digital Signature (496/512)
Reading Digital Signature (504/512)
Read Digital Signature: 4058ef002b3e23c4394d6bdf04b56bae16fda5ff4375fb182
a4807680f065991ed91bce87f236fbb2fdec30c993c13c6d0aea0745970346426f6da7277
28d7d0

        KROOT signature verified!

======== Pending subframes ========
 GST: 1018 432180
 Verified Key 433: 376a812f42c0f13a849b14b74b2e
 Verified Key 469: 2157bbdf7ec1c9da00ea29c6c6fc
 Verified MAC0: d1d6 == d1d6
 Verified SEQ: 00c == 00c
 GST: 1018 432210
 Verified Key 505: a524c630990837b45a69c3b3c1bb
 Verified Key 541: 473da2d109595528aeb1dd4c31ba
 Verified MAC0: bda0 == bda0
 Verified SEQ: 8a8 == 8a8
 GST: 1018 432240
 Verified Key 577: deef6895fb26418c47e26c0ccd52
 Verified Key 613: 946d18a72119aadaf6fc63d2b0e2
 Verified MAC0: ff23 == ff23
 Verified SEQ: d68 == d68
 GST: 1018 432270
 Verified Key 649: 8b27ec5f3a3f08e75e72c5bf482b
 Verified Key 685: 78dfea48975d83a8e5f1d862d802
 Verified MAC0: 85ad == 85ad
 Verified SEQ: 2e5 == 2e5
```

5.1.B Segona part de l'execució: finalitza la lectura de la signatura digital de la clau arrel, la verifica i inicia la verificació de les MACK guardades.



5.1.C Tercera part de l'execució: finalitza la verificació de les MACK guardades, procedeix a verificar la MACK actual i després a llegir un nou *subframe* que resulta ser l'inici del següent missatge DMS i en verifica el camp MACK.

Figura 5.1: Resultat de l'execució de l'escenari 1 del receptor amb `verbose_mack = True`.

Capítol 6

Conclusions

En aquest informe s'ha explicat el funcionament del protocol OSNMA que implementarà Galileo per evitar l'*spoofing* de manera detallada. I s'ha fet una breu introducció als possibles atacs i defenses d'*spoofing*.

S'ha desenvolupat un paquet Python que implementa totes les funcions necessàries per dur a terme les operacions del protocol OSNMA i s'han justificat les decisions de disseny preses alhora de desenvolupar-lo. Pel desenvolupament s'han seguit les normes d'estil oficials de Python PEP8. A més, s'ha publicat el paquet al repositori oficial de paquets PyPI per facilitar la seva descarrega i ús. També s'ha documentat el paquet i s'ha publicat la documentació junt amb el codi públicament on-line.

Per demostrar que el paquet funciona correctament i es viable utilitzar-lo per desenvolupar un receptor, s'ha crear un receptor OSNMA que funciona amb dades de satèl·lit carregades a partir d'un simulador. El receptor funciona només en el cas nominal de transmissió del missatge DMS-KROOT i ofereix informació a l'usuari de les dades que llegeix, les operacions que realitza i també disposa d'opcions de *logging*. El codi del receptor també s'ha fet públic on-line per facilitar la descarrega.

La possibilitat d'implementar un receptor Python fent servir el paquet dissenyat queda validada. Es deixa per futurs projectes la implementació de receptors més realistes i no tan divulgatius que actuïn sobre qualsevol situació del protocol OSNMA. També queda com a futura investigació l'ús del receptor Python que implementa OSNMA per provar l'efectivitat de diferents atacs sobre OSNMA i intentar detectar vulnerabilitats al protocol.

Bibliografia

[Bra07] Georg Brandl, "Sphinx: Python documentation generator", 2007, disponible online a: `https://www.sphinx-doc.org` (accedit al 29 Agost 2020).

[Eij20] Helder Eijs, "pycryptodome 3.9.8", 6 2020, disponible online a: `https://pypi.org/project/pycryptodome/` (accedit al 28 Agost 2020).

[Eur16] European Union, *OSSISICD: Open Service Signal In Space Interface Control Document, Issue 1.3*, 2016.

[Fer19] I. Fernandez-Hernández, T. Ashur, V. Rijmen, C. Sarto, S. Cancela, D. Calle, "Toward an operational navigation message authentication service: Proposal and justification of additional osnma protocol features", *2019 European Navigation Conference (ENC)*, pags. 1–6, 2019.

[FH16a] Ignacio Fernandez-Hernandez, Vincent Rijmen, Gonzalo Seco-Granados, Javier Simón, Irma Rodríguez, J. Calle, "A navigation message authentication proposal for the galileo open service", *Navigation - Journal of The Institute of Navigation*, Vol. 63, 03 2016.

[FH16b] I. Fernández-Hernandez, V. Rijmen, T. Ashur, P. Walker, G. Seco, J. Simón, C. Sarto, D. Burkey, O. Pozzobon, *Galileo Navigation Message Authentication Specification for Signal-In-Space Testing – v1.0*, European Commission, 11 2016, disponible online a: `https://www.gsa.europa.eu/sites/default/files/calls_for_proposals/annex_1-rd4.pdf` (accedit al 29 Agost 2019).

[Fre07] Free Software Foundation, "GNU General Public License, version 3", 2007, disponible online a: `https://www.gnu.org/licenses/gpl-3.0.en.html` (accedit al 29 Agost 2020).

[Goo16] David Goodger, "restructuredtex", 5 2016, disponible online a: `https://docutils.sourceforge.io/rst.html` (accedit al 29 Agost 2020).

[Gri20] Scott Griffiths, "bitstring 3.1.7", 5 2020, disponible online a: `https://pypi.org/project/bitstring/` (accedit al 27 Agost 2020).

[Hum13] T. E. Humphreys, "Detection strategy for cryptographic gnss anti-spoofing", *IEEE Transactions on Aerospace and Electronic Systems*, Vol. 49, nº 2, pags. 1073–1090, 2013.

[Joh17] G. Johnston, A. Riddell, G. Hausler, *The International GNSS Service. Teunissen, Peter J.G., & Montenbruck, O. (Eds.)*, Cham, Switzerland: Springer International Publishing, 1st ed., 2017.

[Mer79] R. C. Merkle, "Method of providing digital signatures", Patent US4309569A, 1979.

[Mis06] P. Misra, P. Enge, *Global Positioning System: Signals, Measurements, and Performance*, Ganga-Jamuna Press, Lincoln MA, 2nd ed., 2006.

[Mol09] A. Molino, M. Nicola, M. Pini, M. Fantino, "N-gene gnss software receiver for acquisition and tracking algorithms validation", *2009 17th European Signal Processing Conference*, pags. 2171–2175, 2009.

[Mot20] Beatrice Motella, Micaela Troglia Gamba, Mario Nicola, "A real-time osnma-ready software receiver", pags. 979–991, 02 2020.

[Nat08] National Institute of Standards and Technology, *FIPS PUB 198-1: The Keyed-Hash Message Authentication Code (HMAC)*, 2008.

[Nat12] National Institute of Standards and Technology, *FIPS PUB 180-4: Secure Hash Standard (SHS)*, 2012.

[Nat13] National Institute of Standards and Technology, *FIPS PUB 186-4: Digital Signature Standard (DSS)*, U.S. Department of Commerce, 2013.

[Nat15] National Institute of Standards and Technology, *FIPS PUB 202: SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*, 2015.

[Per02] Adrian Perrig, Ran Canetti, J. Tygar, Dawn Song, "The tesla broadcast authentication protocol", *RSA CryptoBytes*, Vol. 5, 11 2002.

[Psi16] Mark L. Psiaki, Todd E. Humphreys, "Gnss spoofing and detection.", *Proceedings of the IEEE*, Vol. 104, nº 6, pags. 1258–1270, 2016.

[Ros01] Guido Van Rossum, Barry Warsaw, Nick Coghlan, "Style guide for Python code", PEP 8, 2001, disponible online a: `https://www.python.org/dev/peps/pep-0008/` (accedit al 29 Agost 2020).

[VR09] Guido Van Rossum, Fred L. Drake, *Python 3 Reference Manual*, CreateSpace, Scotts Valley, CA, 2009.

[War20] Brian Warner, "ecdsa 0.16.0", 8 2020, disponible online a: `https://pypi.org/project/ecdsa/` (accedit al 28 Agost 2020).

[Zub18] Xabier Zubizarreta, Johannes van der Merwe, Ivana Lukcin, Alexander Rügamer, Wolfgang Felber, "Receiver independent implementation of the galileo open service navigation message authentication (os-nma)", 11 2018.